Prime numbers

The positive integer n is called **prime** if it has only two divisors: 1 and n.

Representation of the number n in the form $p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$, where p_i are primes, is called **factorization**.

The **fundamental theorem of arithmetic** states that every integer greater than 1 either is a prime number itself or can be represented as the product of prime numbers and moreover, this representation is unique, up to the order of the factors.

For example, $12 = 2^2 * 3$, $100 = 2^2 * 5^2$.

Theorem. Number 1 is neither prime nor composite.

Proof. Number 1 is not composite because it does not have any divisor except itself. If 1 is prime, then for example number 6 has more than one factorization:

$$6 = 2 * 3,$$
$$6 = 1 * 2 * 3,$$

which contradicts the fundamental theorem of arithmetic.

Euclid theorem. The number of primes is infinite.

Proof. Let we have only n primes: $p_1, p_2, \ldots, p_n$. But the number $N = p_1 p_2 * \ldots * p_n + 1$ is not divisible by any of p_i ($1 \le i \le n$) and thus is not composite. We have a contradiction.

E-OLYMP 1616. Prime number? Check if the given number is prime. The number is prime if it has no more than two divisors: 1 and the number itself.

► The number is called ***prime*** if its only factors are 1 and itself.

Theorem. If number n is composite, then it has a divisor no more than $\lfloor \sqrt{n} \rfloor$.

Proof. Let n be composite and d its divisor. Then n / d is also a divisor of n. Assuming that all divisors of n are greater than $\lfloor \sqrt{n} \rfloor$, then $d > \lfloor \sqrt{n} \rfloor$ and $n / d > \lfloor \sqrt{n} \rfloor$. Hence we have $d * (n / d) > \lfloor \sqrt{n} \rfloor * \lfloor \sqrt{n} \rfloor$ or $n > n$. Contradiction.

To test the number n for primality, it is enough to check whether it is divisible by one of the numbers from 2 to $\lfloor \sqrt{n} \rfloor$ inclusive. If n is divisible by at least one of them, then n is ***composite***. Otherwise it is ***prime***.

Function ***IsPrime*** returns 1 if number n is prime and 0 if n is composite.

```
int IsPrime(int n)
{
  for(int i = 2; i <= sqrt(n); i++)
    if (n % i == 0) return 0;
  return 1;
}
```

E-OLYMP 572. The lesson of mathematics Factorize the positive integer *n*. For example, 3240 you must represent in the form 2^3*3^4*5.

► In the problem you need to factorize the number *n*. To do this, sort all its prime divisors from 2 to $\sqrt{n}$, and for each divisor count the number of its occurrences in *n*.

The function ***factor*** factorize the number *n*.

```
void factor(int n)
{
  for(int i = 2; i <= sqrt(n); i++)
  {
    int c = 0;
    if (n % i) continue;
```

Number *i* is a divisor of *n*. In the variable *c* compute the exponent with which it is included in the factorization of *n*.

```
    while(n % i == 0) n /= i, c++;
```

Print the multiple i^c. If *c* = 1, print only the value of *i*.

```
    if (c > 1) printf("%d^%d",i,c); else printf("%d",i);
```

If the number *n* is not factored yet (*n* > 1), print the multiplication sign.

```
    if (n > 1) printf("*");
  }
```

If after the end of the loop the value of *n* is greater than 1, then it is prime.

```
  if (n > 1) printf("%d",n);
  printf("\n");
}
```

Sieve of Eratosthenes is an ancient algorithm for finding all prime numbers up to any given limit. Create a list of consecutive integers from 2 through *n*: (2, 3, 4, ..., *n*). First, numbers greater than 2 and multiples of 2 are removed from the natural series, then numbers greater than 3 and multiples of 3, and so on for each prime number. After the described actions, only prime numbers will remain in the row.

Carry out the described procedure in the array primes[MAX]. First, mark all numbers from 1 to MAX as prime (fill the cells of array primes with 1). Then move through the array from left to right and for each prime number *i*, not greater than $\lfloor\sqrt{\text{MAX}}\rfloor$, mark all numbers of the form *i* * *i*, *i* * (*i* + 1), *i* * (*i* + 2), … as composite (fill the cells with 0).

As a result of executing the ***gen_primes*** procedure, we get

$$\text{primes}[i] = \begin{cases} 1, if\ i\ is\ prime \\ 0, if\ i\ is\ composite \end{cases}$$

```
void gen_primes(void)
{
```

```
  int i, j;
  for(i = 0; i < MAX; i++) primes[i] = 1;
  //primes[0] = primes[1] = 0;
  for(i = 2; i * i < MAX; i++)
    if (primes[i])
      for(j = i * i; j < MAX; j += i) primes[j] = 0;
}
```

Initialize primes array with ones. All integers are declared to be prime.

i	2	3	4	5	6	7	8	9	10	11	12	13	14	15
primes[i]	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Start *for* loop from $i = 2$. primes[2] = 1, so 2 is prime. Start *for j* loop and mark all numbers 2 * 2, 2 * 3, 2 * 4, … as composite.

i	2	3	4	5	6	7	8	9	10	11	12	13	14	15
primes[i]	1	1	0	1	0	1	0	1	0	1	0	1	0	1

Next prime number in the *for* loop is $i = 3$ (primes[3] = 1). Start *for j* loop and mark all numbers 3 * 3, 3 * 4, … as composite.

i	2	3	4	5	6	7	8	9	10	11	12	13	14	15
primes[i]	1	1	0	1	0	1	0	0	0	1	0	1	0	0

In our example MAX = 16, so *for* loop we must iterate till $i = 4$. primes[4] = 0, so 4 is composite. Stop the *for* loop.

The **complexity** of **sieve of Eratosthenes** algorithm is $O(n \log \log n)$.

E-OLYMP **4739. Sieve of Eratosthenes** Given the value of a and b, print all primes in the interval from a to b inclusively.

► Using the Eratosthenes sieve algorithm, fill the primes array, where

- primes[i] = 1, if i is prime;
- primes[i] = 0, if i is composite;

Print all numbers i in the interval from a to b, for which primes[i] = 1.

E-OLYMP **3843. Primes** Let m and n ($2 \le m < n \le 10^7$) be two integers. Consider the following set:

$$\text{Prime}(m, n) = \{ p \mid p \text{ prime}, m \le p \le n \}$$

Find the cardinality of the set Prime(m, n).

► Using the sieve of Eratosthenes, fill the array: primes[i] = 1 if i is prime and primes[i] = 0 otherwise. The size of the primes array is 10^7.

Based on the primes array, fill in the cnt array, where cnt[*i*] contains the number of primes from 1 to *i*:

- if *i* is prime, assign cnt[*i*] = cnt[*i* – 1] + 1;
- if *i* is composite, assign cnt[*i*] = cnt[*i* – 1];

Then the number of primes in the interval [*m*; *n*] equals to cnt[*n*] – cnt[*m* – 1].

The filled arrays **primes** and **cnt** have the form:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
primes[i]	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0
cnt[i]	0	1	2	2	3	3	4	4	4	4	5	5	6	6	6

The number of primes in the interval [4; 12] equals to cnt[12] – cnt[3] = 5 – 2 = 3. These primes are 5, 7 and 11.